

luxury

Freek Wiedijk

Radboud University Nijmegen

2004 11 01, 15:00

two desires

desire 1: a remark by Christine Paulin

- **procedural proofs** ← better for computer science?
Coq, NuPRL, PVS, HOL, Isabelle/tactics
'lists of tactics'
90% of the steps are backward steps
proof scripts unreadable
- **declarative proofs** ← better for mathematics
Mizar, Hyperproof, Tutch, Declare, Isabelle/Isar
'lists of statements'
90% of the steps are forward steps
proof scripts are like mathematical text

desire 2: what mathematicians want

Royal Society discussion meeting in London

‘the nature of mathematical proof’

private talk with Paul J. Cohen

- **formalizations should look like mathematical texts**
‘“type” is computer jargon’
- **formalizations should be developed top-down**
work from axioms
- **proof assistants should hold a dialog**
in case of unclarities, the system should just ask the user

desire 3: saving legacy formal proofs for posterity

John Harrison:

declarative proofs might better survive their system

statements resemble each other between systems



declarative proofs resemble each other between systems

- they will not check (differences in ‘justification’ power)
- they will be useful as a basis for ‘porting’ a proof

but how about existing libraries of procedural proofs?

conversion of procedural proofs to declarative proofs

five ingredients

ingredient 1: Coq-style procedural proof

Lemma Example : forall n, even n -> odd n -> False.

Proof.

induction n.

intros.

inversion H0.

intros.

inversion_clear H.

inversion_clear H0.

auto.

Qed.

n : nat

IHn : even n -> odd n -> False

=====

even (S n) -> odd (S n) -> False

ingredient 2: Mizar-style declarative proof

```
theorem Example: forall n, even n -> odd n -> False
proof
  A1: even 0 -> odd 0 -> False;
  proof assume that even 0 and H0: odd 0;
    thus False by odd_ind,H0;
  end;
  now let n be nat;
    assume IHn: even n -> odd n -> False;
    assume that H: even (S n) and H0: odd (S n);
    H1: odd n by even_ind,H;  H2: even n by odd_ind,H0;
    thus False by IHn,H1,H2;
  end;
  hence thesis by nat_ind,A1;
end;
```

ingredient 3: ALF-style interface

Example =

```
fun n : nat =>
nat_ind (fun n0 : nat => even n0 -> odd n0 -> False)
  (fun (_ : even 0) (H0 : odd 0) =>
    let H1 :=
      match H0 in (odd n) return (n = 0 -> False) with
      | odd_S n H1 =>
        fun H2 : S n = 0 =>
          let H :=
            let H1 :=
              eq_ind (S n)
                (fun e : nat =>
                  match e return Prop with
                  | 0 => False
                  | S _ => True
                end) I 0 H2 in
              False_ind (even n -> False) H1 in
            H H1
          end in
          H1 (refl_equal 0))
    (fun (n0 : nat) (_ : even n0 -> odd n0 -> False) =>
      ?1) n
```

ingredient 4: INRIA-style proof presentation

two ways of generating a natural language proof presentation

- **starting from the lambda term**
works badly with terms from automated tactics
- **starting from the ‘proof tree’**
 - nodes: tactics
 - edges: subgoals

Frédérique Guilhot, Hanane Naciri, Loïc Pottier

compare subgoals with their descendants one level below

different syntax depending on which parts of subgoals are the same

ingredient 5: Mizar-style proof optimization

...

A4: **statement** by A1,A2;

A5: **statement** by A3,A4;

A6: **statement** by A4,A5;

...

- **relprem** will tell you which references are not necessary
- **relinfer** will tell you which statements can be skipped

...

A4: **statement** by A1;

A6: **statement** by A3,A4;

...

declarative proofs from tactics

`'sort | uniq'`

observation: subgoals only change gradually over time

keep every line in the subgoals just **once**

all subgoals can be **merged** into a single declarative proof

proposal: use **incomplete** declarative proofs as proof states

example

now

A1: now

 assume H: even 0;

 assume H0: odd 0;

 thus False by [inversion H0],H,H0;

end;

A2: even 0 -> odd 0 -> False by [intros],A1;

A3: now let n be nat;

 assume IHn: even n -> odd n -> False;

 thus even (S n) -> odd (S n) -> False by [?3],IHn ;

end;

thus forall n, even n -> odd n -> False

 by [induction n],A2,A3;

end;

a framework allowing for both proof styles

- **traditional proof with goals and tactics**

proof = list of tactics

proof state = proof obligation that diminishes until it vanishes

- **alternative proposed here**

proof = both tactics and statements mixed together

proof state = declarative proof that grows until it is complete

two ways of working on a proof:

- manually editing and rechecking the proof (the Mizar way)
- applying a tactic at an unjustified step (the Coq way)

proof trees to proofs: a tiny subset of the Mizar proof language

$\langle \text{proof} \rangle ::= \langle \text{label} \rangle : \mathbf{now}$
 $\langle \text{intro} \rangle^*$
 $\langle \text{proof} \rangle^*$
 thus $\langle \text{statement} \rangle$ **by** $\langle \text{justification} \rangle$;
 end ;

$\langle \text{intro} \rangle ::= \mathbf{let} \langle \text{variable} \rangle \mathbf{be} \langle \text{type} \rangle ;$
 | **assume** $\langle \text{label} \rangle : \langle \text{statement} \rangle ;$

$\langle \text{justification} \rangle ::= \langle \text{tactic} \rangle \{ , \langle \text{label} \rangle \}$

simplification of proofs

example of one of the proof rewriting rules

now		
...		now
now		...
...	→	...
thus statement by ...		thus statement by ...
end;		end;
thus statement by ...		
end;		

prototype

experiment in HOL Light

demo

```
let EXAMPLE = prove
  ('!n. ~(EVEN n) = ODD n',
   INDUCT_TAC THEN ASM_REWRITE_TAC[EVEN; ODD]);;
```

system?

M-mode?

two different approaches to get to a luxurious proof assistant

- **luxurious interface on top of existing proof assistant**

approach used by Isar on top of Isabelle

only practical if you are in charge of the original assistant

Mizar mode

mathematical mode

Mariusz' mode

- **luxurious proof assistant from scratch**

does the world really need **yet another** proof assistant?